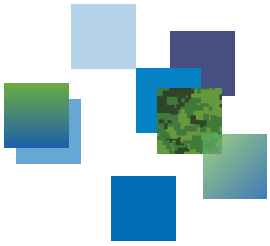
Defence Research and Development Canada
Recherche et développement pour la défense Canada

DRDC | RDDC



Password complexity recommendations

“xez&pAxat8Um” or “P4$$w0rd!!!!”?

Martin Salois
DRDC – Valcartier Research Centre

Defence Research and Development Canada

Scientific Report
DRDC-RDDC-2014-R27
October 2014

Canada

Password complexity recommendations

“xez&pAxat8Um” or “P4$$w0rd!!!!”?

Martin Salois
DRDC – Valcartier Research Centre

Defence Research and Development Canada
Scientific Report
DRDC-RDDC-2014-R27
October 2014

Abstract

The password-cracking world is in turmoil. Many new technologies, such as graphic cards (e.g., NVidia CUDA or ATI/AMD GPUs), make easily available computing power that was previously reserved to very rich organizations. Many efficient tools followed using these technologies, for better or for worse.

This report illustrates the type of performance one can obtain from a generic $2,000 computer and the lessons to learn on password length for different web sites and software programs.

The larger conclusion is that an 11-character password, containing uppercase-lowercase-digits-symbols, is the ideal length for the foreseeable future when no other information is available. More characters are always better; less can be dangerous in the current state of the technology. The use of a password manager, a program that generates and manages strong, complex passwords, is strongly recommended.

Significance for defence and security

Passwords are at the heart of most, if not all, information security measures. As such, it is important that users and system administrators have a minimal understanding of how to choose and manage them. Security guidelines often specify things like minimum length and minimal number of character sets (uppercase, lowercase, digits, symbols), but they never really explain why. This document does exactly that, in layman's terms.

Résumé

Le monde du craquage de mots de passe est en effervescence. Plusieurs nouvelles technologies, comme les cartes graphiques (p. ex. GPU NVidia CUDA ou ATI/AMD), rendent facilement disponible une puissance de calcul qui était auparavant réservée aux très riches organisations. Il s'en est suivi de nombreux produits logiciels hyper-performants qui utilisent ces nouvelles technologies, pour le meilleur et pour le pire.

Ce rapport illustre le type de performance qu'il est possible d'obtenir d'un ordinateur générique de 2 000 $ et les leçons à en retenir sur la longueur des mots de passe pour différents sites Web et types de logiciels comme Microsoft Windows, TrueCrypt et LinkedIn.

La conclusion générale est qu'un mot de passe de 11 caractères contenant majuscules-minuscules-chiffres-symboles est la longueur idéale, jusqu'à preuve du contraire, lorsqu'aucun autre renseignement n'est disponible. Plus de caractères est toujours mieux ; moins peut être dangereux dans l'état actuel de la technologie. L'utilisation d'un gestionnaire de mot de passe, un programme qui génère et gère des mots de passe longs et complexes, est fortement recommandé.

Importance pour la défense et la sécurité

Les mots de passe sont au coeur de la plupart, sinon la totalité, des mesures de sécurité de l'information. Il est donc important que les utilisateurs et les administrateurs de système maîtrisent un minimum de concepts pour bien les choisir et les gérer. Les politiques de sécurité spécifient souvent des choses comme la longueur minimale et le nombre minimal d'ensemble de caractères à utiliser (majuscules, minuscules, chiffres, symboles), mais n'expliquent jamais vraiment pourquoi. Ce document le fait explicitement, en termes faciles à comprendre.

Acknowledgements

Thanks to Sylvain Plamondon, from CGI, for setting up all the tests and benchmarks, and for maintaining all the crazy hacker password cracking tools during the experiment.

This page intentionally left blank.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1 Introduction

Hollywood has caused nearly irreparable damage to user's perception as how passwords are validated. How many times have we seen the computer screen with fast-scrolling characters, with good answers being indicated one by one? This is not a MasterMind game! Password management does not work this way. One either has the whole password validated or nothing at all, and we will see why.

The use of passwords is nothing new. Romans used stone and wood tablets 2,000 years ago to pass the password of the day along to officers. Lost tablets were cause for severe punishment. Passwords were also used during World War II, in the form of a daily challenge/response, to identify friend from foe.

For computer, passwords were first used in 1961 to attribute fair sharing time to users [1]. It was not for security at first. They just wanted to bill the right person and make sure that one user was not monopolizing the computer. Passwords were even stored in the clear! This created problems as unscrupulous users quickly started using other people's credentials.

So, in 1970, Robert Morris came up with an obfuscated representation of the password in the form of a mathematical hash. Believe it or not, this is what is still used today, although the technology has improved to make the hash slower to compute. Yes, you read that right, slower! The problem with the original hash algorithms, and many still commonly used today, is that they were designed for speed, not security, as we will see in this report.

This report starts by explaining what a hash is. It then shows why the speed of hash calculations is a problem and how to actually crack a hash to find the corresponding password. It then presents two sophisticated attacks: pre-calculated tables of hash-password pairs and dictionary attacks. The last two sections combine all the tricks in a small use case and concludes with some recommendations for the users.

2 What is a hash?

A *hash* is a mathematical function that takes a message as input and produces a fixed-length output [2]. One small alteration in the input changes the hash completely. The algorithm is also designed to make it computationally infeasible to find the original message from the hash.

A common hash function is called *MD5*, a version in a series of message digest algorithms designed to ensure the integrity of exchanged messages [3]. Hashes were originally designed with message integrity in mind when network communications

were orders of magnitude slower than they are today. You would send someone an MD5 hash along with a large file or message. That someone could compute the MD5 of what he received and make sure that it matched, thereby ensuring integrity. The idea is that the 128-bit of the MD5 is much smaller than the original message and quick to compute and check. Hashes like these are still in use today. You may have noticed a long string of hexadecimal numbers (0-9ABCDE) next to the download link the last time you downloaded software. This is a hash.

The MD5 function takes an input of any length and, after 64 mathematical iterations, produces an output of 128 bits. Any electronic content can be used, for example:

- The string "password" produces "5F4DCC3B5AA765D61D8327DEB882CF99"
- The 0s and 1s of the executable program for PowerPoint.exe produces "EBBBEF2CCA67822395E24D6E18A3BDF6"
- The Project Guttenberg electronic copy of the French version of the book "The 3 musketeers" produces "85898FEA787421660A990AD89C8049A"

Note that, for convenience, we show a common hexadecimal representation of the 128 its. Otherwise, it would take two lines to represent one hash![1]

If, for example, we change one character in the first one and have "p4ssword" instead of "password", the MD5 hash becomes "93863810133EBEBE6E4C6BBC2A6CE1E7". We can see that the change in the output is rather drastic for such a small change in the input. It was designed this way.

Over the years, as we got more files and larger bandwidth, collision problems started to appear. A *collision* happens when two inputs produce the same hash. Although theoretically very difficult to produce, this started to happen more and more often; enough that it was a concern when used for security. Thus, other more complex algorithms were designed that perform more steps and produce larger hashes.

A well-known improvement on the MD5 is named SHA, for Secure Hash Algorithm, and it comes in a variety of length and complexity [4]. Two common ones are SHA-1 and SHA-512. Both use 80 iterations and produce an output of 160 bits and 512 bits respectively. For example, the SHA-1 of the string "password" is 5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8, while its SHA-512 is:

- B109F3BBBC244EB82441917ED06D618B9008DD09B3BEFD1B5E07394C706A8BB9 80B1D7785E5976EC049B46DF5F1326AF5A2EA6D103FD07C95385FFAB0CACBC86

You can see why these things are not meant to be read and manipulated by humans!

[1]For example, the number 12 in base 10 (what we are used to) is represented as 1100 in binary (base 2) and as C in hexadecimal (base 16). Hexadecimal is more compact and often used for this.

3 The problem of speed

The problem with using these hashes for storing passwords is that these hashes were not designed with security in mind, but rather to improve the speed of communication checks. This means we can crack them faster! Not exactly what we are looking for to securely manage our passwords!

3.1 Cracking a password

So how does one crack a password? Since a hash is one-way—there is no calculation that can give the input (password) given the output (hash)—, the only thing a hacker can do is to try all the passwords from "aaaa" to "ZZZZ". One takes "aaaa", calculates the hash and compares it to the hash he is trying to crack. If it is not the right one, take "aaab", try again and so on. This technique is known as *brute force* cracking. A hacker could manually try all these possibilities, but that would take a very long time. He could try to automate this task, but chances are that even the most basic system will lock you down after a few tries.

The hacker way is to get to the hash or hashes using alternative ways (remote hacking) and then try to crack them offline. For example, a hacker would hack a famous website, get the database of user names and passwords, and then try to crack them locally at his leisure. We will see later that there has been a number of such famous databases publicly released in recent years.

3.2 Setup

To figure out what a bargain-basement hacker could do, we used a $2,000 setup consisting of a Dell Studio XPS 435 [5] computer equipped with an NVidia GeForce GTX 590 EVGA Classified [6]. In a later section, we will provide updated statistics obtained from a much bigger setup.

Why is a Graphical Processing Unit such as this NVidia card so good at cracking password? Parallelism! The GTX 590 has 1,024 cores that can do simple calculations in parallel. Each of these core act as a little processor that can crunch hashes. This technology has taken password cracking from the hands of rich organizations and democratized it for the masses. Unfortunately for us, it didn't take long for hackers to board this train...

3.3 Character sets

How does one calculate cracking speed? Well, it depends mostly on the choice of hash algorithms and what character sets one is using.

There are 4 main character sets that are used:

uppercase (26): ABCDEFGHIJKLMNOPQRSTUVWXYZ

lowercase (26): abcdefghijklmnopqrstuvwxyz

digits (10): 0123456789

symbols (33): č␣!"#$%&'()*+,-./:;<=>?@[\]^_‘{|}~č (where ␣ is a blank space)

There are also other symbols, such as accented characters in language other than English, but they are often unsupported by password tools and are usually not recommended. We will not be using them in this document, but for example:

non standard symbols : ŠŒŽâŤijÂĺÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏâŤIJÃĽÑÒÓÔÕÖØ ÙÚÛÜÝâŤIJÃŮÃčÃĘšœžâŤňÂňâŤňÃĄâŤňâŢŚßàáâãäåæçèéêëìíîïâŤIJâŰŚñ òóôõöøùúûüýÿ ...

Thus, taking the four most common sets yields 95 characters, which is the number we're going to use throughout this report.

3.4 A little math

To calculate the maximum number of calculations required to crack a password, we need the number of characters in our sets as n. We also need to figure out up to what length we want to crack, l. The number of combinations c is then calculated as:

$$c = n^l \tag{1}$$

So, for example, if we set n to the 95 characters we discussed in the previous subsection and want to try all the passwords from 1 to 5 characters ($l = 1$ to 5), we have a total number of 7,820,126,495 combinations to try:

$$c = 95^1 + 95^2 + 95^3 + 95^4 + 95^5 = 7,820,126,495 \tag{2}$$

The time required to try all passwords, t, is simply the number of calculations c divided by the number of passwords we can try per second v:

$$t = c/v \tag{3}$$

Table 1: *Cracking speed for various hash algorithms in tries per second.*

Hash	**No GPU**	**GTX 590** (ratio vs. no GPU)	**GTX Titan x 3** (ratio vs. no GPU, ratio vs. GTX 590)
MD5	131,608,000	3,742,000,000 (~28×)	15,582,000,000 (~118×, ~4×)
SHA-1	17,262,000	1,169,000,000 (~68×)	4,820,000,000 (~279×, ~4×)
SHA-512	1,572,000	161,000,000 (~102×)	292,000,000 (~186×, ~2×)
LM	7,304,000	640,000,000 (~88×)	1,615,000,000 (~221×, ~2.5×)
NTLM	54,575,000	5,645,000,000 (~103.4×)	25,908,000,000 (~475x, ~4.6×)
TrueCrypt	239	3,194 (~13.4×)	7,500 (~31.4×, ~2.3×)

Let's assume for the moment that we can try 3,742,000,000 passwords (or combinations) per second and set v to that number. We'll see why this number makes sense later. This yields the equation:

$$t = 7,820,126,495/3,742,000,000 = 2.09s \tag{4}$$

This means that one can try all the combinations from 1 to 5 characters in about 2 seconds! On average, statistically and for each password length, one needs only to try half the combinations to find the right password. Chances are that, if your password is 5 characters or less, the hacker will get it in one second flat!

3.5 Cracking speed

In the previous subsection, we used $v = 3,742,000,000$ passwords/second as the cracking speed. Where does that number come from? It turns out that this is the speed we achieve with our $2,000 computer setup for MD5 hashes. Remember that the MD5 algorithm is designed for speed, not security. The situation is a bit less dire with more modern algorithms, but clearly other algorithms around the hash itself are required, as we will see. Table 1 shows speed for the algorithms discussed so far plus a few others that will be discussed shortly.

The fourth column in Table 1 is the cracking speed for a much newer ~$8,000 computer

Table 2: *Time required to try all hashes for various password lengths and hash algorithms on the $2,000 computer.*

	Time required (variable units for simplicity)			
Hash	$l = 5$	$l = 10$	$l = 11$	$l = 12$
MD5	2.09 seconds	5 centuries	49 millennia	4.6e+3 millennia
SHA-1	6.69 seconds	1.6 millennia	156 millennia	1.4+4 millennia
SHA-512	48.57 seconds	12 millennia	1.1e3 millennia	1.1e5 millennia
LM	12.22 seconds	*	*	*
NTLM	1.39 seconds	3.4 centuries	32 millennia	3.1e+3 millennia
TrueCrypt	28.34 days	6e+5 millennia	5.7e+7 millennia	5.4e+9 millennia

* LM hashes are a special case (see 3.5.1)

set up with three top-of-the-line (at the time of writing) NVidia GTX Titan cards [7]. The speed here are added for comparison.

One can see that the much bigger and newer SHA-512 algorithm is approximately 23 times slower to crack than its MD5 ancestor. But 161 millions tries per second is still pretty fast! Table 2 shows rounded up numbers for these cracking speed versus time for different password lengths.

The last entry in both tables lists the time for cracking TrueCrypt passwords as an illustration of what a modern algorithm is supposed to do. TrueCrypt [8] is a free disk encryption program designed from the ground up to slow down the cracking process. TrueCrypt offers 24 different combinations of hashes and encryption and it never stores which one was selected by the user. It tries them all to see if the supplied password is correct. Furthermore, depending on the combination, it does between 1,000 and 2,000 iterations of the algorithm. One can clearly see the difference in cracking time versus the password-storage inappropriate MD5!

Looking at those numbers, one might ask "what if I have a 1,000 computers? 10,000? More?". There is the matter of cost, obviously, but what if it's sponsored by a state or organized crime? This is where one has to think of the sweet spot between convenience (what can one remember) and paranoia. In Table 2, we can see that all algorithms take millennias to run at 11 characters. Even if we were to divide that by 10,000 computers, it would still take 4.9 years to crack in the worst case. That is still pretty secure. Thus 11 seems to be the sweet spot. If you are a bit paranoid, add one character (12 total) and even 100,000 computers and the worst hash algorithm would still take 46 years to run!

3.5.1 A side note on Microsoft Windows' LM & NTLM

Table 2 provides the time required to crack LM and NTLM. These are very important hash algorithms as they are used by Microsoft Windows to store user passwords.

The LAN Manager (LM) hash [9] is the legacy algorithm used by versions of Windows prior to Windows NT. Even though we call it a hash, because it is used as such, it is not technically a hash. LM uses DES, an older encryption algorithm. Older encryption algorithms are also extremely fast and, as such, are not a really good way to store user passwords.

The situation is even worse than that! At the time, Microsoft decided that no passwords should ever be longer than 14 characters and they decided to store them in two segments of 7 characters, uppercased! This means that we remove the 26 lowercase characters from our set n, which becomes $n = 95 - 26 = 69$. For a password of 7 characters, this means there are $69^7 = 7,446,353,252,589$ possibilities to try. From Table 1, we can see that we can crack LM hashes at 640,000,000 tries per second. This means that it takes $7,446,353,252,589/640,000,000 = 3.23$ hours to crack every possible LM hashes! This is why Table 2 has no numbers for $l \geq 10$; one never needs to crack anything larger than 7 characters with LM hashes! Unfortunately, even though one would think such an old technology is no longer a problem, because of Microsoft's endless backward compatibility, LM hashes are still stored in newer versions of Windows and it is almost impossible to get rid of. The only real solution is to have password 15-characters and longer. Since those can not be stored in a LM hash, the LM hash is in effect *disabled*. Try telling that to your average users! It is, nonetheless, a very good idea for administrative accounts to have 15 characters or more in their passwords to disable this fast-cracking problem.

The newer NTLM, or NT LAN Manager, uses MD4, an older message digest used before the MD5 we've been discussing. As such, it is not really a good fit for storing passwords either. It can be cracked at the lightning speed of 5,645,000,000 tries per second (Table 1)! This means that, even if one makes up a complex password, made up of all four character sets, if it has 7 characters or less, it can be cracked in less than 4 hours! One needs to get to 10 characters to get a sense of reasonable security.

All that being said, don't get me wrong, this is not Microsoft bashing. Microsoft is the one company that has invested the most in security over the years. They have, however, dropped the ball on password management for some reason.

4 Pre-calculated attacks

As we have seen, cracking by trying out all the possibilities from "aaaa" to "ZZZZ" is quite time-consuming. Hackers realized this early on and have been looking for

ways to accelerate the process. One such way is to pre-calculate all the hashes from "aaaa" to "ZZZZ" in a table and simply look up the hash one wants to crack in that table. Unfortunately for hackers, but lucky for us, this took an impressive amount of hard-drive space that was beyond the average hacker. Notice the past tense. A clever researcher came up with a solution to that problem around 2003: the rainbow tables! More on that later. Let's look at the theory behind pre-calculated tables first.

4.1 Pre-calculated lookup tables

Let's say one wants to crack MD5 hashes. One can do it the hard way (see Subsection 3.1) and try all possible inputs to see if one produces the correct hash. Another way would be to pre-calculate all the possible inputs-outputs and produce a table in which you can just look up the hash. This works because the hashes we discussed so far never change. The MD5 of "password" is and always will be "5F4DCC3B5AA765D61D8327DEB882CF99". There is, however, a problem of storage space.

Let's say we try to produce a file of all possible 3-character passwords. This file would contain one line per input-hash pair. Even if we optimize it, an MD5 stored as a huge number takes up 16 bytes. We need 3 bytes for the input string and usually 2 bytes for the end-of-line characters. This means $16+3+2 = 21$ bytes per line. Since $n = 95$, we will have $n^3 = 857,375$ lines times 21 bytes for a total of $18,004,875$ bytes, or 17.2 megabytes. Not so bad, but remember this is only for 3-character passwords that would not take very long to crack in the first place; not much of an improvement.

If we try it for 5-characters passwords, we get 165.7 gigabytes—almost 10,000 times bigger than for 3 characters. For 7-character passwords, this creates a 1.6 petabyte table—a petabyte is 1,048,576 gigabytes or 1,024 terabytes—, almost 100,000,000 times bigger than for 3 characters and 10,000 times bigger than for 5 characters. As you can see, it is exponential. It multiplies by approximately 95 (n) every time you add one character, give or take the few extra bytes per line.

1.6 petabytes, even by today's standard of $0.03 per gigabytes would cost a whopping $50,340 [10]! Ten years ago, that was unthinkable! At $2.58 per gigabyte, that would have cost over 4 million dollars. But 10 years ago is when the clever researcher we mentioned came up with rainbow tables.

4.2 Rainbow tables

The name of the researcher is Philippe Oechslin and his seminal paper [11] was published in 2003. In it, he highlights a time versus memory trade-off to pre-calculated

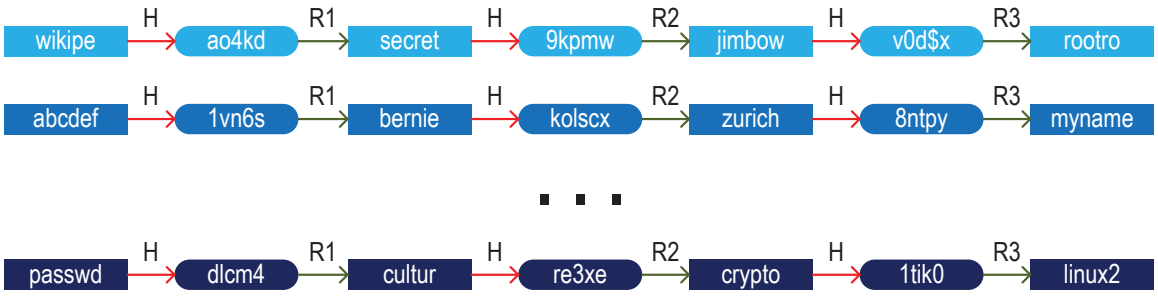


Figure 1: *Rainbow chains.*

tables. The details of how rainbow tables precisely work is beyond the scope of this document. Here is the short version.

Oechlin came up with the concept of reduction functions. A reduction function takes as input a hash and produces another input string. Keep in mind that a reduction function is not a hash and may not have the same properties. Chaining these reduction functions, you get a list of input-hash-input-hash-...-input-hash-input. You then create as many such chains of a determined length as you wish to trade time for space. You can create different reduction functions or reuse the same ones. Conceptually, it looks like what is shown in Figure 1—each chain is usually represented in a different color, which might explain the name *rainbow*. The beauty of the algorithm is that all the chains can be recomputed given only the starting and end inputs, which are the only ones recorded on disk, hence saving an incredible amount of space.

When you want to crack a hash, you start in reverse. Let's say we want to find the password for the hash "re3xe". Now we can see it at the bottom of Table 1, but remember that this part of the chains does not get saved. So we need to find which chain contains our hash.

We start with the right-most reduction function R3 and apply it to the hash. This gives "rambow". Is "rambow" the end input of any chain? No. We then try with the right-most two reduction functions. We apply R2 to the hash. This gives "crypto". Then we compute the hash for "crypto", which gives "1tik0". We then apply R3 to "1tik0", which gives "linux2". Is "linux2" the end input of any chain? Yes. It is the end of the last chain in our rainbow table. We now know that this is the right chain. We need to rebuild it until we get to our hash and the last input is the password we are looking for: "cultur".

Don't worry if this is not perfectly clear. As mentioned, this is the bare minimum version. The important thing to remember is that rainbow tables are a very condensed pre-calculated table that have allowed hackers access to your 8-or-less-character passwords for $n = 95$ for the weak hash algorithms we have seen so far. That is, until the

smaller tables became obsolete with the advent of powerful home GPUs about half a decade ago. Another important thing to know is that rainbow tables are useless against modern hashing algorithms who use a simle trick: salts!

4.3 Defeating pre-calculations with salt

Remember that the MD5 of "password" is and always will be "5F4DCC3B5AA765D61D8327DEB882CF99"? This greatly reduces the time required to crack a database of user-password pairs since, once you have found one hash, everyone with the same hash has the same password. It also means that you can use pre-calculated tables, of the full or rainbow variety.

There is a very easy solution to this problem. This solution is used by every self-respecting and modern hash algorithms. It is called a salt and it is simply a predetermined number of random bytes that are added to your password. Let's say we use two bytes, chosen at random for each user every time they change their password. For example, Alice's salt is "AB" while Bob's salt is "z9". Even if both use "password" as a password, their hashes are no longer the same:

- Alice's hash: MD5("ABpassword") = 755BA76CC6269D36989CE5E27F064DBA
- Bob's hash: MD5("z9password") = CD6B8276A151ECE0769AE047D8F57F1D

This "trick" has been used in Linux's password management for a very long time, along with some other tricks that make cracking much longer, although not impossible except for much longer passwords. If you have ever seen a shadow file, where user/hash pairs are stored in Linux, you may have seen user entries that look like this:

```
root:$6$aWYFiEfl$86tn6L2q.nYxIEXVB.dcVcottcvceDyKfJxqXj2DYeq0.cwXW7s
 L10M3cEOnT5xEBZKZhoym8DriiOc4Stbyc1:15902:0:99999:7:::=
```

The text in red is the user name. The text in dark green is the type of hash for this particular entry. 6 in this case means SHA-512. The text in blue is the salt. Finally, the text in purple is the hash, encoded with a custom base64 algorithm. The algorithm used by default in most Linux version does 5,000 iterations, each of which does something different with the password-salt pair and the results of the previous iterations. For comparison, on our $2,000 computer, we do just over 19,000 tries per second, much slower than the 161,000,000 unsalted and plain SHA-512 shown in Table 1, but still faster than for TrueCrypt.

5 Dictionaries and other tricks

Up to this point, we have assumed the worst possible case for the hacker: a complex, long and random password containing characters from all four main sets (uppercase, lowercase, digits, symbols). Unfortunately, this is not how most users will select a password. Human nature has conditioned us to use memory tricks and make our life as easy as possible. This is why the average user will use words from his vocabulary with, perhaps, if we're lucky, some capitalization, numbers and symbols.

This has lead to a simple technique known as a dictionary attack. In this type of attack, the hacker tries all the words in a dictionary. This can be the English dictionary, any dictionary customized for a particular target, or a dictionary of all the previous passwords the hacker ever found by himself or online. The hacker can also try combinations of two or more passwords.

The idea is that this creates a much smaller search space to go through then a brute force attack combining all the characters from the four sets. For comparison, the English Merriam-Webster dictionary has only 476,000 words. Many linguists indicate that a dictionary of common spoken English words has about 150,000 entries. Using that to try all possible 3-word combinations, we get $150,000^3$ =3.376e+15. An impressive number for sure, but still orders of magnitude smaller than a brute force attack for up to 15 characters: 95^{15} =4.6e+29!. It is also surprisingly good at finding passwords, as we will see with leaked password databases.

5.1 Leaked dictionaries and user databases

The last five years have seen a huge increase in hacking user databases. As hackers have realized the potential of a good dictionary in cracking passwords, they have been actively seeking and sharing their dictionaries. This trend arguably started with the RockYou database.

RockYou is a videogame company with millions of online users. In December 2009, hackers got access to its user database and posted it online [12]. The problem is that the password were stored in clear text, no hash or encryption! This allowed all sorts of statistical analysis to be made. The database contained 32 million users, with a little over 14 million unique passwords. Table 3 presents the 20 most popular passwords from that list along with their frequency.

As one can see, users are not very imaginative and mostly do not seem to care. So-called keyboard walks are common: "qwerty", "123456", "q1w2e3r4t5", etc. First names are also very popular; so is using the name of the website—rockyou in this case—or simple variations.

Table 3: *RockYou's top 20 passwords, with their frequency.*

Rank	Password	Frequency	Rank	Password	Frequency
1	123456	290,731	11	nicole	17,168
2	12345	79,078	12	daniel	16,409
3	123456789	76,790	13	babygirl	16,094
4	password	61,958	14	monkey	15,294
5	iloveyou	51,622	15	jessica	15,162
6	princess	35,231	16	lovely	14,950
7	rockyou	22,588	17	michael	14,898
8	1234567	21,726	18	ashley	14,329
9	12345678	20,553	19	654321	13,984
10	abc123	17,542	20	qwerty	13,856

Many other such databases have been leaked in recent years [13], such as LinkedIn (6.46 millions users), MySpace, Sony, and the biggest one yet, Adobe [14] (56+ millions entries). Looking at all these databases, it seems that "password" and "123456" are clear winners!

A lot of users might argue that these are unimportant passwords and that is why they chose easy-to-remember passwords. Security analysis over the years have shown however that this is not true. Most users reuse the same passwords for everything. In the case of RockYou, for example, this might lead an industrious hacker, armed with a username and password, from RockYou to Gmail to FaceBook to your credit card account; all because the user is reusing passwords. In the case of a business or the government, this can lead to access to privileged information if the hacker successfully poses as you to your boss or colleagues.

There are further problems with these leaked password. Not only will they help recover identical passwords in the future, but they also provide good training grounds for statistical and user behaviour analyses.

5.2 Statistical models

Some of the most useful statistical analyses that can be extracted from leaked user-passwords databases are Markov models [15] of letter, or group of letters, frequency and positioning. The complete theory behind those models is far beyond the scope of this document. Suffice to say that they provide answers to questions such as: what letter often comes after this letter?; what letter comes more frequently at position i?; what group of two letters often come after this letter or at position i?

Table 4: *Common tricks devised by users to generate their passwords, using "password" as an example, loosely ordered by statistical occurrence.*

Trick	**Example(s)**
Capitalize	Password
Uppercase	PASSWORD
Leet-speak	p@$$w0rd
Reverse	dwowssap
Suffix numbers	password0, password1, ..., password9, ... password2014
Suffix symbol	password!
Prefix numbers	0password, 1password, ..., 9password, ... 2014password
Remove vowels	psswrd
Repeat	passwordpassword
Truncate	passw

This can provide guidance for an hybrid brute force attack where the most probable passwords come before the most improbable ones. This can significantly reduce the time required for a brute force attack. This can also help in identifying the most common "tricks" that users employ in devising their password.

5.2.1 Common user tricks

Everyone thinks they have found a perfect trick to create their password. What the analysis of the leaked databases has shown is that a lot of people have the same tricks!

As we have seen, many users will just use a word they know and/or a series of numbers. But the next class of users, who are just a bit more security-aware, will complexify their password a bit with some easy-to-remember substitutions and/or transformations.

One such substitution is the so-called *leet speak*. Common in cell phone text messages and chat programs, this is a dialect in which common letters are replaced by other similar-looking characters. For example, replace the letter "o" by the digit "0"; "a" by "4" or "@"; "e" by "3"; etc. Using this technique, for example, "password" can become "p@$$w0rd".

Table 4 is a list of leet speak and other common tricks that will be tried by hackers. If your trick is in the list, please change it!

Another trick is to use two or three of those tricks. For example, add numbers and symbol: "password2014!". A common trick that has been observed, and one that is

the default for one of the most efficient cracking tools, is this:

ULLL...LLSDDDDS

where U, L, S and D stand for uppercase, lowercase, symbol and digit respectively; the four common sets we've been talking about. This says that most users, when required to use all four sets, will start with an uppercase letter, followed by some lowercase letters, 0 or 1 symbol followed by 1 to 4 digits and a final symbol. For example, this can crack a password of the form "Password-2013.". The idea, once again is to greatly reduce the search space. To find "Password-2013." without this trick, one has to try $95^{14} = 4.9\text{e}{+27}$ combinations. Using this trick, one "only" has to search through $26 \times 26^7 \times 33 \times 10^4 \times 33 = 2.3\text{e}{+18}$. Once again, this is a huge number for sure, but it is 9 orders of magnitude smaller!

And, once again, if that was your trick, please change it!

6 Combining all the tricks

As time has shown, hackers can be quite clever in their mischievousness. Knowing all the tricks mentioned so far, they still strive to do even better. What they are doing right now is combining those tricks to find the ultimate solution [16] and customize it to the time they can spend cracking a particular database or user password. This allows them to find passwords that are still recommended as secure by many experts and guidelines. The article gives "qeadzcwrsfxv1331" as an example, but there are many others.

Looking at the literature and our own experience, here is a sample tested methodology that could give us between 50 and 90% of the passwords in a large database within 24 hours. We will use the numbers for MD5 cracking on our $2,000 computer. Remember from Table 1 that we can crack those at 3,742,000,000 tries per second.

1. Brute force 1 to 6 characters in 3.31 minutes ($n = 95$).
2. Use a dictionary like RockYou or a combination of other leaked databases in a few minutes, plain and with some of the common tricks.
3. Brute force from 7 to 9 lowercase letters in 25 minutes ($n = 26$).
4. Brute force from 7 to 13 numbers in 50 minutes ($n = 10$).
5. Combine dictionary + brute force up to 4 numbers in a few minutes ($n = 10$). For example, "word2014".
6. Brute force 7-character passwords in 5.25 hours ($n = 95$).

7. For the remaining time, use the markov model, built from leaked databases or passwords found so far if enough were found, to brute force 8 characters and up.

8. Give up after 24 hours! Chances are that the remaining passwords are truly random and very long. You'll never find them!

This is assuming that there is no enforced policy on password length and complexity. If that is the case, adjust the methodology accordingly. For example, if a password must have a minimum of 8 characters, with at least one uppercase and a number, then steps 1, 3, 4 and 6 should be skipped or adjusted.

7 Conclusion and recommendations

The moral of this story is that you don't want to be in the easy-to-crack password lists. Just as you probably don't want to leave your home's door unlocked so that anyone can get in, there is a level of risk you must always assume, but you want to minimize it. There are $10-locks and $1,000-locks; there are good-enough passwords that you can actually remember and very secure, extra-long, random passwords that you can't remember.

The idea is to find a good compromise that works for you and for the level of security you want for a particular application. For a one-time registration on a web site, it's probably ok to use a simple password. But don't ever reuse that password for your bank account or anything with actual personal information like Gmail and FaceBook. These information-rich sites and programs should have their own secure and unique passwords.

Also know that, in 2014, there is absolutely no reason for an application or a website to limit the length or complexity of your password. Be wary of such limits as they may indicate a lack of understanding of basic security concepts from the developers.

If you have a very good memory for abstract characters, a random password like "xez&pAxat8Um" will always be better than "P4$$w0rd!!!!" for the same length. If your memory is not so good, make it longer. Adding lots of symbols at the end is easy and will defeat most of the techniques described here. So, for example, why not add an extra five "!" at the end (e.g., "P4$$w0rd!!!!!!!!!")?

Whatever you do, please stay away from the tricks we have seen and use characters from all four sets (uppercase, lowercase, digits, symbols). In terms of length, we have seen that the sweet spot seems to be 11 characters at the time of writing. Add one character if you are feeling paranoid or for very important sites. For administrator accounts in Microsoft Windows, remember to have at least 15 characters.

As a final note, there's hope for those of use with no memory at all. There are password managers that can manage all of that for you. As we cannot endorse a particular product, look for "password managers" in Google or within your existing security products, such as antivirus programs. Many of them have started offering the service. Such a program will generate extra long, random passwords, often 32 characters or more. All you need to remember is the one master password to access the software. Better make this one an extra long and complex one!

References

[1] Password (online), http://en.wikipedia.org/wiki/Password (Access Date: 2014-04-02).

[2] Hash function (online), http://en.wikipedia.org/wiki/Hash_function (Access Date: 2014-04-02).

[3] MD5 (online), http://en.wikipedia.org/wiki/MD5 (Access Date: 2014-04-02).

[4] Secure Hash Algorithm (online), http://en.wikipedia.org/wiki/Secure_Hash_Algorithm (Access Date: 2014-04-02).

[5] Dell Studio XPS 435 Desktop (online), http://www.dell.com/us/dfh/p/studio-xps-435/pd (Access Date: 2014-04-02).

[6] NVidia GeForce GTX 590 (online), http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-590 (Access Date: 2014-04-01).

[7] NVidia GeForce GTX Titan (online), http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan (Access Date: 2014-04-01).

[8] TrueCrypt — Free Open-Source On-The-Fly Disk Encryption Software for Windows 7/Vista/XP, Mac OS X and Linux (online), http://www.truecrypt.org (Access Date: 2014-04-01).

[9] LM Hash (online), http://en.wikipedia.org/wiki/LM_hash (Access Date: 2014-04-01).

[10] Komorowski, M., a history of storage cost (online), http://www.mkomo.com/cost-per-gigabyte (Access Date: 2014-04-01).

[11] Oechslin, P. (2003), Making a Faster Cryptanalytic Time-Memory Trade-Off, In *Advances in Cryptology: Proceedings of CRYPTO 2003, 23rd Annual International Cryptology Conference. Lecture Notes in Computer Science*, Santa Barbara, California, USA: Springer. ISBNÂă3-540-40674-3.

[12] Cubrilovic, N. (December 2009), RockYou Hack: From Bad To Worse (online), http://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords (Access Date: 2014-04-02).

[13] Lystad, A., Leaked passwords (online), http://thepasswordproject.com/leaked_password_lists_and_dictionaries (Access Date: 2014-04-02).

[14] Ducklin, P. (November 2013), Anatomy of a password disaster — Adobe's giant-sized cryptographic blunder (online), nakedsecurity (Access Date: 2014-04-02).

[15] Markov model (online), http://en.wikipedia.org/wiki/Markov_model (Access Date: 2014-04-02).

[16] Goodin, D. (May 2013), Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331" (online), http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords (Access Date: 2014-04-02).

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated.)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DRDC – Valcartier Research Centre
2459 de la Bravoure Road, Québec QC G3J 1X5, Canada

2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC APRIL 2011

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Password complexity recommendations: “xez&pAxat8Um” or “P4$$w0rd!!!!”?

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Salois, M.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
October 2014	30	16

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Scientific Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

DRDC – Valcartier Research Centre
2459 de la Bravoure Road, Québec QC G3J 1X5, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
05AA	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC-RDDC-2014-R27	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

The password-cracking world is in turmoil. Many new technologies, such as graphic cards (e.g., NVidia CUDA or ATI/AMD GPUs), make easily available computing power that was previously reserved to very rich organizations. Many efficient tools followed using these technologies, for better or for worse.

This report illustrates the type of performance one can obtain from a generic $2,000 computer and the lessons to learn on password length for different web sites and software programs.

The larger conclusion is that an 11-character password, containing uppercase-lowercase-digits-symbols, is the ideal length for the foreseeable future when no other information is available. More characters are always better; less can be dangerous in the current state of the technology. The use of a password manager, a program that generates and manages strong, complex passwords, is strongly recommended.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

password management and complexity; public cryptography and hash

DRDC | RDDC

SCIENCE, TECHNOLOGY AND KNOWLEDGE FOR CANADA'S DEFENCE AND SECURITY | SCIENCE, TECHNOLOGIE ET SAVOIR POUR LA DÉFENSE ET LA SÉCURITÉ DU CANADA

www.drdc-rddc.gc.ca